**eda\_analyze.py**

There are two main types of analysis this module manages:

1. **Event-Related Analysis (eda\_eventrelated)**:
   * This is used when you have short chunks of data (called "epochs") that are time-locked to specific events.
   * For example, you might want to analyze the EDA responses in the 5 seconds immediately after a participant sees a scary picture versus a neutral picture.
2. **Interval-Related Analysis (eda\_intervalrelated)**:
   * This is for longer, continuous recordings.
   * For example, a 5-minute resting-state session or an entire experiment.

**Libraries Importing:**

**from .eda\_eventrelated import eda\_eventrelated**

**from .eda\_intervalrelated import eda\_intervalrelated**

**Function Definition:**

**def eda\_analyze(data, sampling\_rate=1000, methods="auto"):**

* **data**: The main input.
* **sampling\_rate=1000**
* **methods="auto"**: e.g., event\_related or interval\_related.

method = method.lower()

* Converts the method string to all lowercase letters.

**Logic Branches:**

**Path 1: Event-Related Analysis**

# Event-related analysis

if method in ["event-related", "event", "epoch"]:

* This checks if you explicitly asked for event-related analysis.

# Sanity checks

if isinstance(data, dict):

for i in data:

columns = data[i].columns.values

elif isinstance(data, pd.DataFrame):

columns = data.columns.values

* This part just gets the list of column names from your data, regardless of whether your data is a dictionary or a single DataFrame.

if len([i for i in column if "label" in i]) == 0:

raise ValueError(...)

* **Label**: Event-related analysis needs to know which event each data point belongs to. The standard way to store this information is in a column named "label".

features = eda\_eventrelated(data)

* If the "label" column is found, the function hands the data over to the eda\_eventrelated specialist, which does the actual calculations and returns the results in the features variables.

**Path 2: Interval-Related Analysis**

# Interval-related analysis

elif method in ["interval-related", "interval", "resting-state"]:

features = eda\_intervalrelated(data,

sampling\_rate=sampling\_rate)

* This branch is taken if you explicitly ask for interval-related analysis. It's simpler because it doesn't need a "label" column. It just passes the data and sampling rate to the eda\_intervalrelated specialist to calculate the overall statistics.

Path 3: Automatic Detection

elif method in ["auto"]:

* This is the "smart" mode where the manager has to decide for itself.

if isinstance(data, dict):

for i in data:

duration = len(data[i]) / sampling\_rate

if duration >= 10:

features = eda\_intervalrelated(data, sampling\_rate=sampling\_rate)

else:

features = eda\_eventrelated(data)

* **if isinstance(data, dict):**: It first checks if the data is a dictionary (which is common for epochs).
* **duration = len(data[i]) / sampling\_rate**: Calculates the duration of the data chunk being checked.
* **if duration >= 10:**: If a data chunk is 10 seconds or longer, it assumes it's an interval and uses eda\_intervalrelated. Otherwise, it uses eda\_eventrelated.

if isinstance(data, pd.DataFrame):

* This handles the case where the data is a single DataFrame.

if "label" in data.columns:

epoch\_len = data["label"].value\_counts()[0]

duration = epoch\_len / sampling\_rate

* It first checks for the "label" column. If it exists, it assumes the data is a set of epochs.
* **data["label"].value\_counts()[0]**: This is a clever way to find out the length of a typical epoch by counting how many rows belong to the most common label.
* **duration = epoch\_len / sampling\_rate**: It calculates the duration of that epoch.

else:

duration = len(data) / sampling\_rate

* If there is no "label" column, it assumes the entire DataFrame is one long recording.
* **duration = len(data) / sampling\_rate**: It calculates the total duration of the entire DataFrame.

if duration >= 10:

features = eda\_intervalrelated(data, sampling\_rate=sampling\_rate)

else:

features = eda\_eventrelated(data)

* It applies the same 10-second rule as before to choose the correct specialist function.

return features

* Finally, it returns the features, based on the condition.